УДК 004.021

РАЗРАБОТКА И РЕАЛИЗАЦИЯ PARALLEL-BATCHED ДЕРЕВА ИНТЕРПОЛЯЦИОННОГО ПОИСКА

Кокорин И.В. (Национальный исследовательский университет ИТМО), **Марценюк А.Р.** (Национальный исследовательский университет МФТИ),
Научный руководитель – к.т.н, доцент ИТиП Аксёнов В.Е.
(Национальный исследовательский университет ИТМО)

Структура данных называется parallel-batched если она может за один запрос вставить, удалить или найти сразу набор ключей и использует параллелизм для ускорения. В ходе этой работы мы разработали parallel-batched дерево интерполяционного поиска, которое отвечает на запросы из гладких распределений асимптотически быстрее, чем широко известные деревья поиска, и использует параллелизм для более быстрого исполнения запросов.

Введение. Упорядоченное множество является одним из самых часто используемых в программировании абстрактных типов данных. Существует множество возможных реализаций упорядоченного множества с использованием различных структур данных: упорядоченных массивов, деревьев поиска, списков с пропусками.

Большинство реализаций упорядоченного множества поддерживает только *скалярные* запросы: за один такой запрос можно вставить (или удалить, или найти) в множество единственный ключ. Но помимо скалярных операций структура данных может поддерживать *батч-запросы*: за один такой запрос можно выполнить операцию вставки, удаления или поиска сразу для множества ключей.

Исполнение батч-запросов на современных многоядерных компьютерах может быть ускорено с помощью параллельного программирования. Структуры данных, которые параллелизуют исполнение батч-запросов, и таким образом добиваются более быстрого их исполнения, называются parallel-batched структурами данных.

Существует несколько parallel-bached реализаций упорядоченного множества на основе 2-3 деревьев, красно-чёрных деревьев, AVL-деревьев, декартовых деревьев и т.д. Тем не менее, не существует parallel-batched реализаций упорядоченного множества, которые при определённых условиях (например, гладкости распределения ключей) работают асимптотически быстрее обыкновенных сбалансированных деревьев поиска.

В этой работе мы реализуем parallel-batched дерево интерполяционного поиска, которое способно отвечать на запросы из гладких распределений за время O(log log Size), что асимптотически быстрее O(log Size) — времени, за которое на такие запросы отвечает обычное сбалансированное дерево поиска.

Основная часть. Дерево интерполяционного поиска (interpolation search tree, IST) — структура данных, реализующая интерфейс упорядоченного множества. Каждая вершина IST содержит *представителей* — массив из m упорядоченных по возрастанию ключей k1, k2… km и, если эта вершина не лист, массив указателей на m+1 ребёнка. При этом ключи, содержащиеся в поддереве первого ребёнка, строго меньше k1; ключи, содержащиеся в поддереве i-ого ребёнка находятся в интервале (k_{i - 1}, ki); а ключи, содержащиеся в поддереве m+1-ого ребёнка, строго больше km. Заметим, что поддерево каждой вершины IST также будет являться IST.

Кроме того, каждая вершина IST содержит легковесный *индекс*, позволяющий с помощью интерполяции за константное время определять примерное местоположение искомого ключа в массиве *представителей*.

Дерево интерполяционного поиска, содержащее N ключей, называется *идеальным*, если его корень содержит O(N^(1/2)) ключей, каждая вершина на втором уровне содержит O(N^(1/4)) ключей, каждая вершина на третьем уровне содержит O(N^(1/8)) ключей и так далее. Нетрудно доказать, что идеальное IST будет содержать O(log log N) уровней. Если

распределение хранящихся в IST ключей гладкое, то с помощью *интерполяционного индекса* мы сможем в каждой вершине за константное время определить, где продолжать поиск искомого ключа. Таким образом, для поиска ключа в IST нужно посетить O(log log N) вершин, в каждой вершине совершив O(1) операций. Таким образом, для гладких распределений идеальное IST может отвечать на запросы за O(log log N) операций, что асимптотически быстрее O(log Size) — времени, за которое на запросы отвечает обычное сбалансированное дерево поиска.

Parallel-batched IST будет исполнять каждую операцию (вставка, удаление, поиск) над упорядоченным массивом ключей. В вершине V для каждого обрабатываемого ключа нужно параллельно определить, в какого ребёнка V нужно пойти для продолжения обработки этого ключа. После этого разные поддеревья V могут быть обработаны параллельно: каждое поддерево V будет независимо и параллельно обрабатывать непрерывный подмассив исходного массива ключей.

Для того, чтобы поддерживать сбалансированность parallel-batched IST, нужно реализовать два примитива

- build параллельно строит идеальное IST из упорядоченного массива ключей
- flatten параллельно преобразует произвольное IST в упорядоченный массив ключей, хранящихся в нём

Таким образом, для преобразования произвольного IST в идеальное мы сможем использовать комбинацию flatten и build: сначала необходимо собрать ключи, хранящиеся в преобразуемом IST, в упорядоченный массив, а после построить идеальное IST из этого упорядоченного массива.

В каждой вершине V мы будем подсчитывать число внесённых в поддерево V изменений (удалений и вставок). После того, как число этих изменений, умноженное на константу c, превышает исходный размер поддерева V, поддерево V целиком перестраивается и преобразуется в идеальное IST. Преобразование поддеревьев, в которые было внесено слишком много изменений, нужно для того, чтобы изменения не приводили к слишком сильной разбалансировке IST, сохраняя его близким к идеальному.

Найдя при проходе по IST, в массиве представителей вершины V, ключ, который необходимо удалить, мы не удаляем ключ из массива представителей, а просто помечаем ключ как удалённый — удалённые ключи не будут возвращаться операцией flatten.

Ключи, которые необходимо вставить в IST, будут вставляться либо в произвольное поддерево при его перестройке, либо в один из листьев, если за весь спуск ни одно поддерево на пути не было перестроено.

Выводы. В ходе этой работы нами была разработана реализация parallel-batched упорядоченного множества на основе дерева интерполяционного поиска. Разработанная реализация отвечает на запросы из гладких распределений за O(log log Size) и испольует параллелизм для ускорения *батч-запросов*. Разработанная реализация может использоваться для хранения данных в СУБД в случаях, если нужно поддерживать быструю обработку множества ключей за один запрос.

Кокорин И.В. (автор) Подпись

Аксёнов В.Е. (научный руководитель) Подпись